USF UNIVERSITY OF SAN FRANCISCO

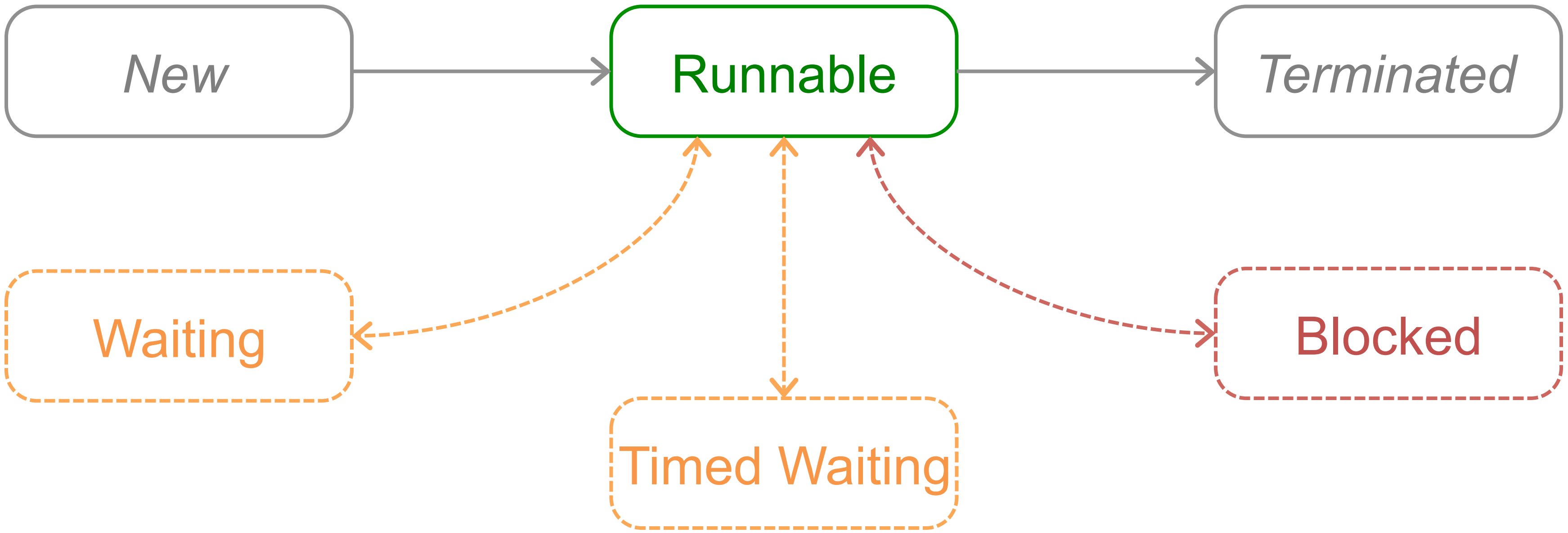
CHANGE THE WORLD FROM HERE

Multithreading

Thread Pools and Work Queues

Chris Bryan ◆ cbryan2@usfca.edu
Department of Computer Science

Thread States



http://docs.oracle.com/javase/8/docs/api/java/lang/Thread.State.html

Motivation

- **Goal**: Web Server
  - Must handle multiple simultaneous requests
  - Must be **responsive** AND **efficient** (e.g. respond quickly, finish quickly)
- **Implementation**: Multithreading
  - One thread per request?

Problems

- Overhead cost to **creating objects**
  - Initialization in constructor (and super() calls)
- Overhead cost to **destroying objects**
  - Garbage collection
- Overhead cost to **excessive memory usage**
  - Causes thrashing

Solutions

- Keep Threads Around
  - Initialize a "wise" number of threads once
  - Reuse threads for other tasks instead of destroying
- Two Approaches
  - Thread pool
  - Work queue

Thread Pools

- Create a fixed number of worker threads
- When have work to do…
  - Get available thread from pool and assign task
  - Thread runs assigned task
  - Thread returns to pool of available threads
- What if there are no available threads?

Work Queue

- Add a work queue to thread pool
  - (And don't block the caller!)
- Threads check for available work in queue
  - Usually remove work in FIFO fashion
  - If no work, thread waits until queue is not empty
- When have work to do…
  - Add work to queue and return

Keeping Threads Around

- **Thread Pools**
  - Basically an array of threads that sticks around
  - Simple, but causes blocking
- **Work Queues**
  - Adds a queue of "work" (runnable objects)
  - More complicated, but responsive

Potential pitfalls

- **Deadlock** – all threads executing tasks that are currently blocked waiting for the results of another task on the queue, but the other task cannot run because there is no unoccupied thread available.
- **Resource thrashing** – too many threads eating up system resources
- **Concurrency Errors** – Incorrect `notify()` and `wait()` calls leading to missed notifications
- **Thread leakage** – a thread is removed from the pool to perform a task, but is not returned to the pool when the task completes, such as via unhandled `Exceptions`

IBM Work Queue

Java Theory and Practice:
Thread Pools and Work Queues
IBM developerWorks
http://www.ibm.com/developerworks/library/j-jtp0730/index.html

Chris Bryan ◆ cbryan2@usfca.edu
Department of Computer Science
UNIVERSITY OF SAN FRANCISCO

USF UNIVERSITY OF SAN FRANCISCO

CHANGE THE WORLD FROM HERE

Chris Bryan ◆ cbryan2@usfca.edu
Department of Computer Science